Team 11 Final project

***Abstract*— Robotic arms, originating from the days of assembly automation, have undergone significant study and expansion. Originally confined to factories, modern robotic arms now boast cutting-edge computer vision systems and precise control mechanisms, enabling their application in diverse scenarios. In this paper, Team 11 presents the successful development of a robotic arm capable of autonomous ball sorting. The robotic arm pictured in Figure 1 boasts 4 degrees of freedom and a spherical end-effector capable of gripping objects about a hand's width apart. To achieve an autonomous pick and place robot, Team 11 experimented with various kinematic approaches include forward and inverse kinematics, cubic and quintic trajectory generation, and various image processing techniques, and more. The implementation leverages powerful software tools including Matlab's computer vision apps and the Dynamixel SDK to read in camera data and process the images, enabling the accurate identification and isolation balls for sorting. In addition to the balls, the robot is able to correctly identify other small objects by color and move them to their binned location.**

I. INTRODUCTION

Worcester Polytechnic Institute's Unified Robotics III course covers the basic principles surrounding conventional, serial robotic manipulators. For the final project for this course, Team 11 worked with a 4-Degree of Freedom (DoF) RRRR OpenManipulator-X arm with a simple servo-actuated gripper at the end-effector. The 4 Dynamixel motors positioned at each joint allowed for both joint and task space control, enabling the use of both forward, inverse, and differential kinematics. Additionally, the use of a webcam, a black and white checkered board, and Matlab's camera calibration and image processing toolboxes allowed for complex computer vision tasks including image aquisition, RGB masking, median filtering, centroid mapping, and object depth localization. This paper explores a 7 week process of understanding kinematics and image processing tasks in order to create an autonomous pick-and-place system. The robot was tasked with identifying, isolating, picking, and binning small, lightweight, 3D-printed balls of 5 different colors. The robot was expected to use a quintic trajectory control function to ensure a smooth path during movement. It was also expected to run on a Ubuntu 20.04 LTS operating system and communicate via a UBS communication protocol. Team members were expected to collaboratively code via Github.

i

II. METHODS

Before using a robotic arm, it is imperative to be able to know where the robot is and how it is oriented. The process to figure this out is called Forward Kinematics and using this process with our 4-DoF robot proved to be a multi-step endeavor. To start calculating the forward Kinematics the lab group sketched out the reference frame at each joint shown in figure 1.

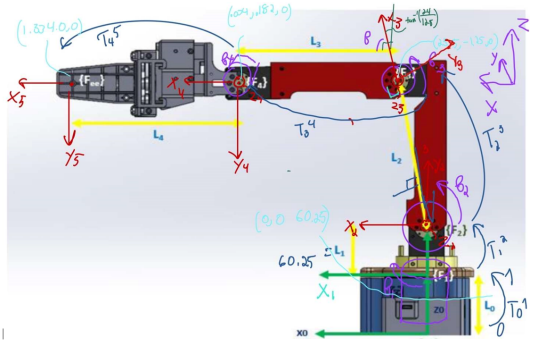

Fig. 1. skech of robotic arm with intermediate reference frames drawn

θ	d	a	α
0	L_0	0	0
θ_1^*	L_1	0	$-\frac{\pi}{2}$
$\theta_2^* + \left(\arctan\left(\frac{24}{128}\right)\right) - \frac{\pi}{2}$	0	L_2	0
$\theta_3^* - \left(\arctan\left(\frac{24}{128}\right)\right) + \frac{\pi}{2}$	0	L_3	0
θ_4^*	0	L_0	0

TABLE I. table of DH parameters derived from robotic arm using skech in Fig. 1

This sketch was used to determine the DH parameters for the robot shown in table II.The DH parameters could then be substituted to the general definition of a homogeneous transformation matrix Equation n to get the transformation for that specific joint. Multiplying all the transformation matrices together yielded the transformation matrix that described the location of the end effector with respect to the global origin shown in equation II.

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta)\cos(\alpha) & -\sin(\theta)\sin(\alpha) & a\cos(\theta) \\ \sin(\theta) & \cos(\theta)\cos(\alpha) & -\cos(\theta)\sin(\alpha) & a\sin(\theta) \\ 0 & \sin(\alpha) & \cos(\alpha) & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

These transformations would be vital for all future calculations. With the position of the end effector with respect to joint angles (Forward Kinematics) calculated, the next step

was to calculate the joint angles with respect to a given end effector position. Because this is a 4-DoF robot, we are able to control for 4 degrees of freedom or X position, Y position, Z position, and one orientation angle. In the case of the robot used in the lab, the angle of the end effector in the X-Z plane, called alpha, is able to be controlled.

While solving for the inverse kinematics of a robotics manipulator, there are two popular methods: the algebraic approach and the geometric approach. The group chose to pursue the geometric approach because the sketches could fit on a normal page- unlike the math for the algebraic approach. To begin, the group drew a simple sketch of the robot in 3d space with clear labels and a color code for all joints and links, shown in Figure 2.

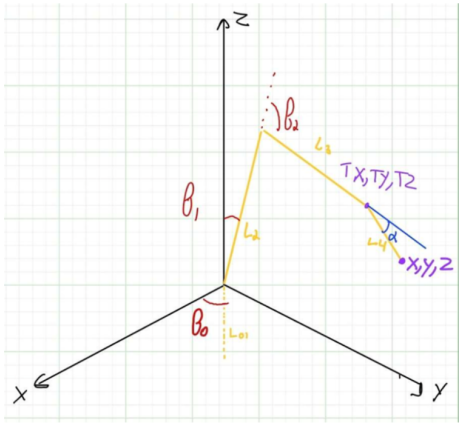

Fig. 2. initial sketch of robot arm used in geometric approach

The sketch can be simplified by removing link 0 and 1 because its length contributes to the Z component regardless of joint orientation. With a neat and concise sketch, we moved on to breaking the links up into solvable triangles, shown in Figure 3.

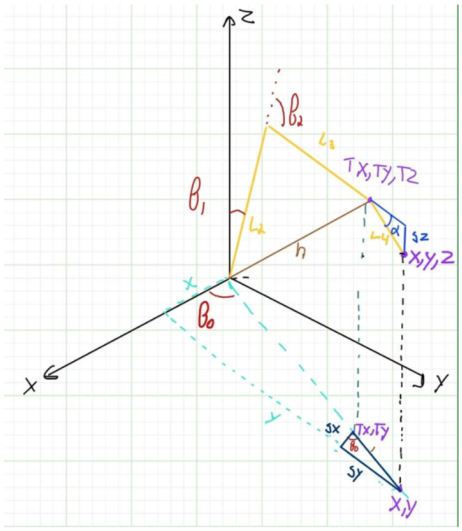

Fig. 3. simple sketch of robot arm with projections included

Starting with the simplest terms, θ_0 is calculated by projecting our end effector into the XY plane as shown in the dotted turquoise triangle in Figure 3.

$$\theta_0 = \arctan(\frac{X}{Y}) \tag{2}$$

If link 4 is removed from the equation, θ_1 and θ_2 can be solved using the law of cosines. The challenge was to associate the XYZ location of the end effector with the transitional point T_x, T_y, T_z. This is solved by drawing a right triangle on L_4 with alpha. The triangle's height gave us the difference between Z and T_z shown in equation 3.

$$T_z = (Z - L_0^1 + \sin(a) + L_4) \tag{3}$$

The base of the triangle could be projected to the XY plane with θ_0 to form another triangle drawn in navy blue on Figure 3. Using this triangle T_x and T_y were easily found using equations 4 and II, where $L_4 \cos(a)$ was the base of the triangle consisting of link 4 and α.

$$T_x = (X - \cos(\theta_0) L_4 \cos(a)) \tag{4}$$

$$T_y = (Y - \sin(\theta_0) L_4 \cos(a)) \tag{5}$$

With the position of the end of link 3 defined as T_x T_y and T_z, we created another triangle spanning link 2 and 3 called h, shown in figure 4.

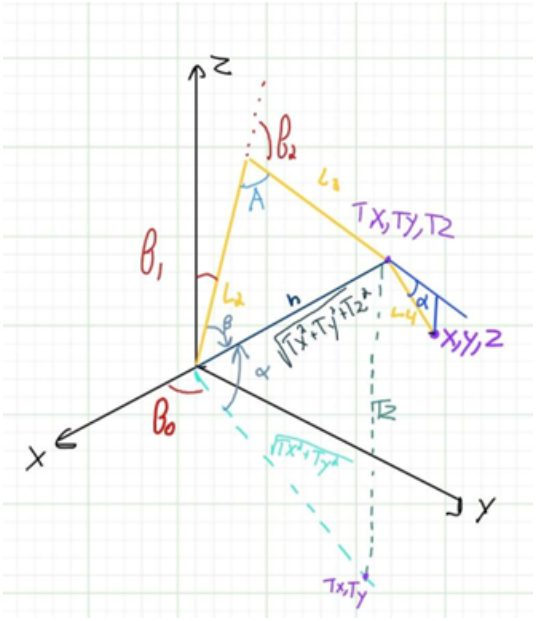

Fig. 4. simple sketch of robot arm with projections included

The length of h was found using the Pythagorean theorem of T_x T_y and T_z. Our next step was to calculate θ_1 based off α and β. α, shown in equation 6, was found using simple trigonometry.

$$\alpha = \alpha \tan(\frac{T_z}{\sqrt{T_x^2 + T_y^2 + T_z^2}} \tag{6}$$

$$\beta = \arccos(\frac{L_2^2 + T_x^2 + T_y^2 + T_z^2 - L_3^2}{2L_2\sqrt{T_x^2 + T_y^2 + T_z^2}} \quad (7)$$

Finally, α and β can be combined to obtain θ_1, but the DH table's offsets must be factored in to keep all reference frames consistent. The DH table specified the offset for θ_1 as shown in equation 8.

$$\theta_1^* + \arctan(\frac{24}{128}) - \frac{\pi}{2} \quad (8)$$

So the inverse inverse is added to solve for θ_1.

$$\theta_1^* = \frac{\pi}{2} - (\beta - \alpha) - \arctan(\frac{24}{128}) \quad (9)$$

θ_2 could be solved in the same manner; the law of cosines gives angle A shown in figure 4, and then the inverse is added to the offsets to solve for θ_2.

$$\theta_2^* - \arctan(\frac{24}{128}) + \frac{\pi}{2} \quad (10)$$

$$\theta_2^* = \arccos(-\frac{L_2^2 + L_3^2 - T_x^2 - T_y^2 - T_z^2}{2L_2 * L_3}) - \frac{\pi}{2} + \arctan(\frac{24}{128}) \quad (11)$$

With θ_0, θ_1, and θ_2 found, deriving θ_3 just required finding the difference between their sum and α.

$$\theta_3^* = \alpha - T_1 - T_2 \quad (12)$$

The full inverse kinematic equations are shown below:

$$\theta_1^* = \frac{\pi}{2} - (\beta - \alpha) - \arctan\left(\frac{24}{128}\right)$$

$$\theta_2^* = \arccos\left(-\frac{L_2^2 + L_3^2 - T_x^2 - T_y^2 - T_z^2}{2L_2 \cdot L_3}\right) - \frac{\pi}{2} + \arctan\left(\frac{24}{128}\right)$$

$$\theta_3^* = \alpha - T_1 - T_2 \quad (13)$$

For the team's manipulator to effectively collect and sort the colored balls the robot needs to be able to observe its environment. To that end, a camera was employed to achieve this task. At the very start, the camera calibrates itself using many pictures of the checkerboard from different angles to remove distortion from its fisheye lens as seen in Figures 5 and 6. Now that the robot can calibrate itself to undistort images the robot can finally observe its environment taking a picture of its surroundings.

There is one problem, however, this image includes data that should not be included in object detection. Considering this the pixel coordinates of the edges of the checkerboard were logged and a poly mask was applied to eliminate everything in the image except the checkerboard and items on it, this makes the next step much easier as no colors in the background will be present. Now that the robot can only see the checkerboard, color thresholding was employed

Fig. 5. Image of Fisheye Capture

Fig. 6. Example of Calibration Image

to isolate the ball of a given color, effectively masking out the entire checkerboard and the other non-relevant colored balls. Attempts were made at using a mask that bounded and labelled all the balls in one image, but was ultimately less efficient. Masking per color allowed for greater threshold accuracy and easier scalability when writing code. The result of this is a binary image of a colored ball of the robot's choice, this allows the robot to differentiate between each color of ball.

Though in theory, the ball should be isolated, this binary image has noise which produces error in generating the centroids of a given ball. To remove this noise a median filter was applied to each image resulting in only the selected ball to be shown. Now that the ball has been isolated, bounded, and labelled, its centroid is determined in pixel coordinates. These pixel coordinates, however, are not sufficient to command the robot. It is required to convert these pixel coordinates to checker space coordinates, and then to robot space coordinates.

However, another problem must be dealt with before converting from checker to robot space. This is the perspective error. Since the camera is finding the centroids in 2D pixel space, when projecting to convert from pixel to checker space we are projecting through the ball which would log a point behind it. To compensate for this we applied geometric calculations which can be seen in equations 14 and 15 these calculations compensate for the perspective projection in the x and y directions.

$$X_{corrected} = X + \sin(\arctan(\frac{X_{camera} - X_{ball}}{Y_{camera} - Y_{ball}})) * ball_{height} * \frac{\sqrt{(X_{camera} - X_{ball})^2 + (Y_{camera} - Y_{ball})^2}}{camera_{height}} \quad (14)$$

$$Y_{corrected} = Y + \cos(\arctan(\frac{X_{camera} - X_{ball}}{Y_{camera} - Y_{ball}})) * ball_{height} * \frac{\sqrt{(X_{camera} - X_{ball})^2 + (Y_{camera} - Y_{ball})^2}}{camera_{height}} \quad (15)$$

Finally, now that the checker space ball location has been compensated for we can convert the location to robot space coordinates, allowing us to command the robot to the exact location of the ball only using the camera.

CODE

Over the course of this project, team members developed an extensive code base.

Among these classes include files to control: camera, arm and joint paths, graph plotting, trajectory generation, individual color masks, tester files, and more.

Each class has a function in the autonomous system. Some like the model class are mostly for auxilliary functions such as generating plots. Others like the arm class read in the hardware and send it along the serial lines to the software house.

Others are more pertinent to the project.

The robot class is responsible for controlling the arm. It is capable of reading joint variables, setting joint variables, manipulating the gripper, completing the forward kinematics through methods such as dh2mat and dh2fk and inverse kinematics through methods such as $ik_algowhichrunstheNewton - Raphsonalgorithm.ItalsocallsontheTraj_plannerclasstorunthecubicandquintictrajectorymethods.$

The camera class calibrates the camera position and mathematical values based on example images. This file is responsible for taking into account the lens, lighting, positioning, and rotation of the camera.

On the other hand, each of the createMask files were generated using the Matlab toolbox. They respond to the RGB threshold of each color ball and filter out any values that are not present within this filter.

The intitialization is contained in the lab5 tester file. The file begins with initializing the camera file to calibrate the camera. It then uses the camera classes pointsToWorld and the matlab camera class to turn the image file into a mathematical plane. The next step is to mask the image so that the balls can be clearly identified and so that the individual color mask classes can be called. Once the balls are isolated, the centroids are established and the median filter is placed on each image.

Finally, the state machine is revealed in the Final Project class. The machine travels from looking for the balls (taking in the image, checkerboard, masking), to looking for each individual ball color, moving to each ball using the robot.m's arm and gripper functions. Once each ball color is found, the centroid is mapped to pixel coordinates and assigned a binning location. Within the moving state, both the trajectory and gripper classes are called to ensure the robot picks and places as desired in keeping with the object localization.

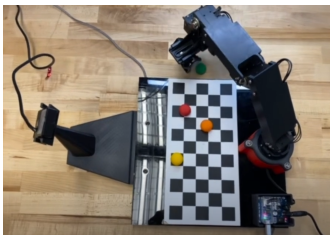

Fig. 7. Unsorted Balls

Fig. 8. Sorted Balls

III. RESULTS

As the code for the robot progressed certain computations proved to be more or less intensive than others. Inverse kinematics was the first computation to have a noticeable impact on the robot's performance; though once this was reformatted into a symbolic form this computational load dropped significantly. Later, we found trajectory planning to be difficult due to the large amount of computations required. While we reduced the frequency of these computations as much as possible, the robot's movements often remained jerky. More optimization should be completed in order to make these trajectories useful.

Though the robot was sometimes jerky, overall it proved to be very accurately controlled. When sent to a specific location its error was small enough to be acceptable. This could be seen in the final pick and place routine where the robot was able to line up with small balls with no noticeable error every time.

In the end as seen in Figures 7 and 8, the robot was able to accurately find, map out, pick-up, and place at a rate of 20 seconds per ball.

The robot was able to correctly identify red, green, yellow, orange, and gray balls and place them in a location based on color. It was also able to understand when the ball had been picked up and the endeavor had been unsuccessful based on the torque read-in by the grippers. It was able to go after missed balls without re-calculating the whole image.It was also able to pick up items that were not shaped as the balls including markers and gift packets. The robot was an overall success.

IV. DISCUSSION

Though not simple, the creation of this pick-and-place mechanism highlighted the versatility of the methodologies used. Any robotic manipulator can benefit from the use of forward kinematics to know where it is as well as inverse kinematics. These tools do more than prepare the robot

for the desired task, they prepare it for any task. Once a robot's unique homogeneous transformation matrices and its symbolic form of the Jacobian is figured out, any task within its range of motion and strength becomes an ease. Our group spent a fraction of the time preparing the robot for a pick-and-place routine than we did solving the necessary equations for forward and inverse kinematics. Should we want to change its task, we could do so easily given the proper sensors.

This ease of application allows for robotic manipulators to become mass producible. While robotics within niche fields are cool, and important for pushing the field forward; a large portion of the industry is made up of robots that can do many different things.. This is to say that the same robotic arm may be sold to a car manufacturer and to an amusement park. While this is largely due to shared mechanical criteria, the ease at which a robot's application can be changed is what truly allows this to happen. Companies like New Bedford Research and Robotics spend much of their time training robots to do various tasks for various companies, showing the importance of robotic manipulator versatility in the real world.

V. CONCLUSIONS

To conclude this project, Team 11 successfully developed an autonomous pick-and-place robotic system using a variety of kinematic and image processing techniques. Team members learned how accurately to manipulate position, velocity, and acceleration in joint and task space while ensuring the robot stayed within a reachable workspace using standard techniques such as DH conventions, Jacobian matrices, cubic and quintic trajectories, Newton-Raphson algorithms, and more.

The team ran into several challenges over the last few weeks. Among them, the quintic trajectory function was so severely unoptimized, the robot's oscillations were destroying any hope for accuracy. Reducing the number of calculations allowed the arm to work smoothly.

Team members also ventured into the realm of computer vision using Matlab's powerful libraries. The incorporation of computer vision techniques, including image acquisition, RGB masking, median filtering, centroid mapping, and object depth localization, presented Team 11 with the challenge of creating a sophisticated yet efficient sorting mechanism for small, multicolored, 3D-printed balls. By experimenting with a variety of masking functions, the team was able to increase the speed of the robot's picking path, accuracy of the centroid placement, and the scalability of the autonomous identification.

To complete this project, the team utilized a number of industry-standard tools including running Github on Linux, Dynamixel motor software, Matlab computer vision libraries, and more.

The team would like to acknowledge the support of their incredible teacher's and TA's in making this project a reality.

SECTION	AUTHOR
Introduction	Hudson/Amber/Prakriti
Methodology	Hudson/ Dylan
Results	Amber/Prakriti
Discussion	Amber
Conclusion	Prakriti
Code Review	Prakriti

TABLE II. Table of contributions

VI. TABLE OF CONTRIBUTIONS

All other pertaining sections including video-making, code planning, experimentation, documentation, and write-ups were split evenly between the team members listed above. Video was made though iMovie present on team member's Hudson's devices. Team member Hudson was also active in path planning on this project. Team member Dylan was instrumental to understanding camera calibration while team member Amber was very active in understanding the object localization math. Team member Prakriti spent some time experimenting with masking varieties.

APPENDIX

Demo Video or go to the url: `https://youtu.be/Vd1bfmvgXD8`

Link to Codebase or go to the url: `https://github.com/RBE3001-C24/RBE3001_C24_Team11/tree/main`